Z zaskoczeniem możemy odnotować fakt, że panująca do niedawna wśród profesjonalistów moda na stosowanie mikroprocesora 8080 lub co najwyżej Z80 jakby nieco osłabła. Konstruktorzy przymierzają się i do innych mikroprocesorów — chociaż w próbach z „16-bitowcami" monopol firmy INTEL utrzymuje się niezmiennie.

Tym, co koniecznie potrzebują podparcia w fakcie, że dany układ produkowany jest w którymś z krajów RWPG spieszymy donieść, że znana do niedawna z wypasu owiec Bułgaria podjęła produkcję bogatej rodziny układów z serii 6800.

Opisany w artykule moduł został zaprojektowany tak, aby mógł współpracować z zasobami (pamięć, układy we-wy) systemu przystosowanego do sterowania przez 8080. Pozwala to na stosunkowo „bezkrwawe" udoskonalenie posiadanego systemu. Możliwa jest też do realizacji konstrukcja wykorzystująca dwa procesory np. Z80 i właśnie 6800 (przełączane), podobnie jak spotyka się to w niektórych rozwiązaniach komputerów kompatybilnych z APPLE II. Tym, którzy chcieliby uzyskać szczegółowe informacje podajemy kontakt telefoniczny z p. Jerzym Orkiszewskim: tel. domowy 21-36-20, tel. do pracy 49-98-71 w. 538. Autor dysponuje również pewnymi modułami programowymi dla systemów wykorzystujących mikroprocesor 6800.

Moduł procesora MOTOROLA 6800

Mikroprocesor MC6800 (U1 — rys. 1) ma 16-bitową szynę adresową, 8-bitową synchroniczną szynę danych oraz trzy podstawowe sygnały sterujące: E (ang. Enable) — zegar systemu, który synchronizuje przesłania na szynie danych, R-W/ — (ang. Read-Write) — wskazujący kierunek przesyłania danych oraz VMA (ang. Valid Memory Address) sygnalizujący poziomem wysokim H obecność na szynie adresowej stabilnego adresu pamięci. Mikroprocesor 6800, podobnie jak 8080, wymaga zewnętrznego układu zegarowego. Rolę tę spełnia układ MC6875 (U2), który dostarcza sygnały zegarowe Φ1 i Φ2 o poziomach MOS, przyjmuje sygnał wstrzymania MRDY (odpowiednik WAIT/) oraz żądania DMA lub cyklu odświeżenia pamięci dynamicznej DMA/-REF REQ/. W przypadku braku aktywnego sygnału MRDY (poziom L) układ 6875 zatrzymuje przebiegi zegarowe w stanie Φ1=„0", Φ2=„1" co odpowiada drugiej połowie cyklu zegara (rys. 2). Reakcją układu na sygnał DMA/-REF REQ/ jest zatrzymanie zegara w pierwszej połowie cyklu oraz ustawienie linii REF GRNT w stan „1". Oscylator wymaga dołączenia kwarcu o częstotliwości czterokrotnie większej niż częstotliwość zegara systemu, która dla standardowej wersji elementów rodziny M6800 wynosi 1 MHz. Warto dodać, że w związku dynamiczną realizacją wewnętrznych rejestrów mikroprocesora częstotliwość ta nie może być niższa niż 100 kHz.

Układ 6875 wytwarza ponadto następujące sygnały: E, który z uwagi na dużą obciążalność wyjścia nie wymaga buforowania (N=30 TTL), sygnały pochodzące z oscylatora 4f i 2f oraz sygnał MCLK (ang. Memory Clock). MCLK daje identyczny przebieg jak E — jednak nie reaguje na stan wejść wstrzymujących, a zatem może być pomocny przy obsłudze pamięci dynamicznej.

Buforowanie sygnałów systemowych

Wydajność prądowa wyjść mikroprocesora pozwala na bezpośrednie wysterowanie ok. 10 układów należących do rodziny M6800, co odpowiada jednemu standardowemu obciążeniu TTL. W wypadku dołączania do systemu typowych bloków pamięci niezbędne są wzmacniacze. Szyna adresowa buforowana jest przez układy U3 i U4 (74LS244) a szyna danych — układ U5 (74LS645). Pozostałe sygnały buforowane są bramkami 74125. Układ 74LS244 jest jednokierunkowym, trójstanowym nadajnikiem szyny (ang. octal bus driver). Układy te mogą być zastąpione przez inne o podobnej funkcji; należy jednak uważać, aby nie przekroczyć obciążalności wyjść mikroprocesora A10...A15, które sterują dwoma układami: U3 i U6. Nadajnik-odbiornik szyny 74LS645 (ang. octal bus transceiver) może być zastąpiony dwoma krajowymi układami UCY 74S416.

Bufory szyny adresowej, danych i linii R-W/ wprowadzane są w stan dużej impedancji przez buforowany sygnał BA (ang. Bus Available). Mikroprocesor wymusza na linii BA poziom aktywny (H) w odpowiedzi na wysterowanie poziomem aktywnym (L) wejścia HALT/ lub wykonanie rozkazu WAI — „czekaj na przerwanie". Kierunek transmisji przez układ U5 wyznacza poziom linii R-W/. Bufory U11 i U12 (74125) są otwarte na stałe (z wyjątkiem bufora linii R-W/).

Zerowanie systemu

Procesor zostaje wyzerowany przez podanie poziomu niskiego (trwającego przynajmniej osiem taktów zegara) na wejście RESET!. Źródłem tego sygnału może być albo układ 6875, realizujący funkcję „power-on reset" (ang. zerowanie przy włączeniu zasilania), lub inny układ zewnętrzny wymuszający logiczne zero na linii RESIN/. W celu zachowania kompatybilności z układami rodziny MCS80 wyprowadzony jest dodatkowo sygnał RESET, aktywny przy poziomie wysokim. Układ zerowania zbudowany został na bramkach układu U13 (7400).

Sygnały sterujące typu INTEL

Do układu jednostki centralnej dołączony został moduł wytwarzający sygnały sterujące, zgodne z szyną rodziny INTELA. Pozwala to na wykorzystanie wcześniej skonstruowanych bloków np. pamięci, we-wy lub specjalizowanych sterowników, np. napędów dysków elastycznych lub monitora telewizyjnego. Moduł zrealizowany został z układów U6 (74LS138), 1/2 U7 (74S139), 1/4 U13 (7400) oraz U8 (74S416).

Mikroprocesor 6800 nie ma rozkazów we-wy takich, jak IN i OUT dla mikroprocesora 8080. Układy we-wy traktowane są jako adresy pewnego bloku w obszarze adresowym. W celu wygenerowania sygnałów I-OR/ i I-OW/ przyjęto, że układy we-wy umieszczone będą w obszarze pamięci o rozmiarach 1 KB. W celu zasygnalizowania, że mikroprocesor generuje adresy z tego obszaru, wytwarzany jest sygnał PA/. Powstaje on w dekoderze U6 (74LS138) sterowanym liniami adresowymi A10...A15 w taki sposób, że w zależności od położenia przełącznika P sygnał PA/ generowany jest dla adresów z zakresu 8000_H-$83FF_H$ lub 9000_H-$93FF_H$,...,FOOO-F3FF. Sygnał PA/ doprowadzony jest do dekodera U7, do którego przyłączony jest również sygnał R-W/. W zależności od stanu tych linii aktywne są wyjścia I-OR/, I-OW/ bądź MEMR/, MEMW/. Pierwsze przyjmują stan aktywny dla PA/=L, drugie dla PA/=H. Dekoder uaktywniany jest tylko w drugiej połowie okresu zegara (E przyjmuje poziom H), o ile procesor generuje odwołanie do pamięci (VMA przyjmuje poziom H/).

Należy zwrócić uwagę, że sygnał PA/ użyty do wydzielenia obszaru adresowego dla układów peryferyjnych rodziny 6800 musi być ziloczynowany z sygnałem VMA. Można to zrealizować w sposób, w jaki zaprojektowano dekoder adresowy dla układu MC6850. Tworzą go U9 i 1/2 U7.

Intelowskie sygnały sterujące buforowane są układem 74S416, wprowadzanym w stan dużej impedancji sygnałem BA.

Sygnały przerwań

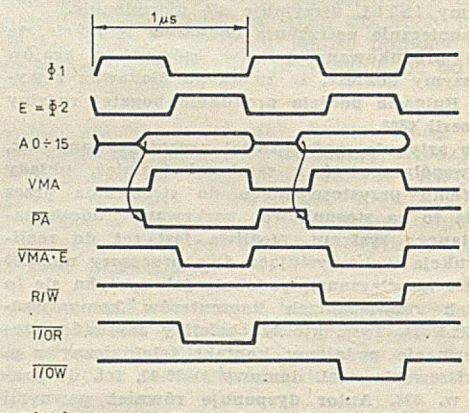
Mikroprocesor 6800 przyjmuje dwa przerwania zewnętrzne — maskowalne IRQ/ i niemaskowalne NMI/. Na oba te wejścia, jeżeli nie są używane, powinien być podawany poziom wysoki.

Układ we-wy szeregowego MC6850

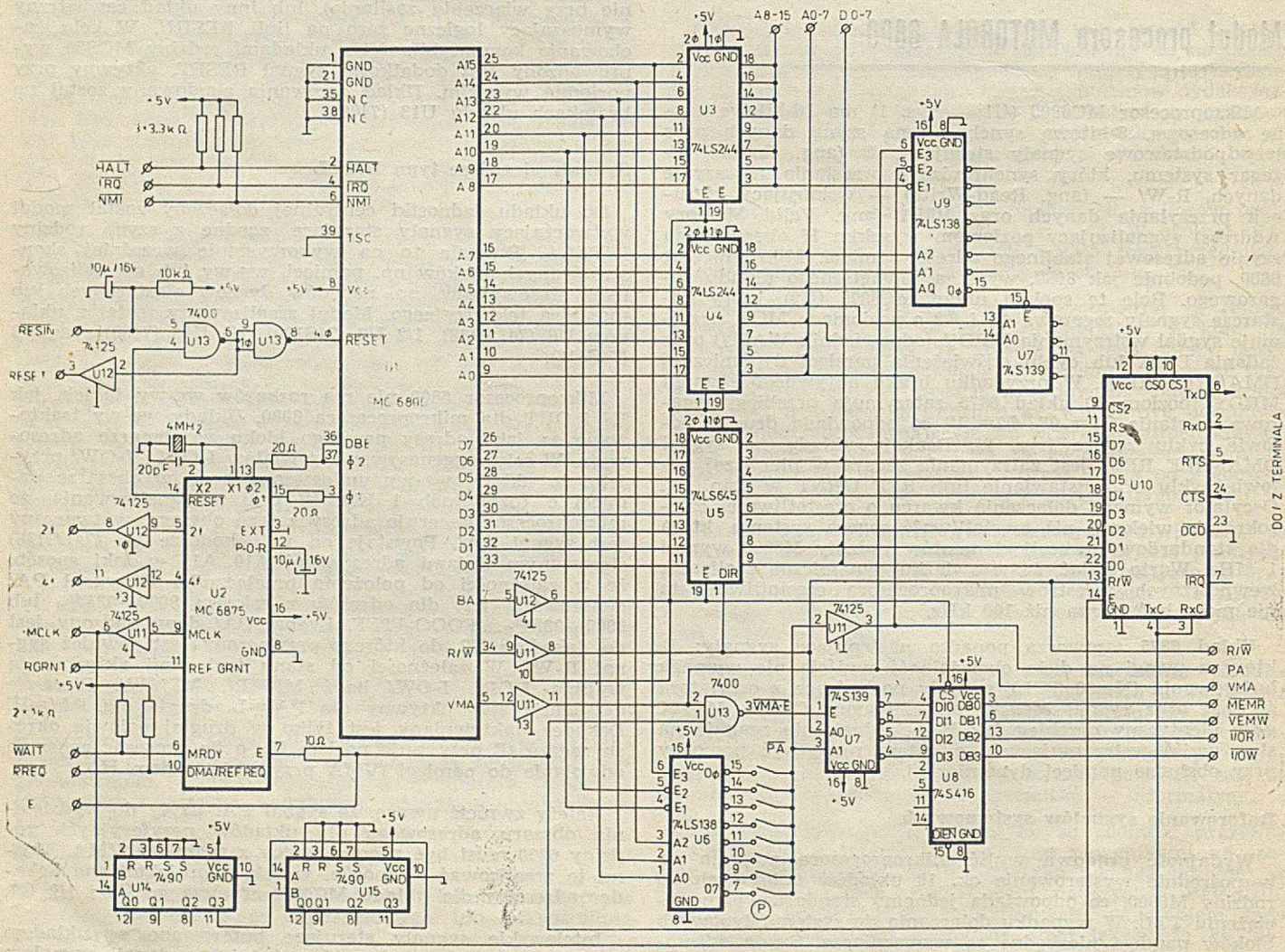
Do układu CPU dołączony został układ 6850, znany pod nazwą Asynchronous Communications Interface Adapter — ACIA (ang. adapter asynchronicznego interfejsu komunikacyjnego). Z pewnym przybliżeniem można powiedzieć, że spełnia on rolę analogiczną do 8251. Obecność tego układu w zaprojektowanym systemie wynika z faktu, że jest on często wykorzystywany przez procedury we-wy bardzo bogatego oprogramowania firmowego dla rodziny 6800. Układ może zostać wykorzystany do współpracy z terminalem lub innym komputerem.

Oznaczenia wyprowadzeń układu od strony systemu są zgodne z opisanymi liniami mikroprocesora, a od strony terminala odpowiadają oznaczeniom linii wg standardu RS-232C. Warto przypomnieć, że linia CTS/ oznacza gotowość terminala do przyjęcia znaku i jej niski poziom na wejściu 6850 odblokowuje sekcję nadajnika. Linia RTS/ sterowana jest programowo. Wyjście sygnału przerwania IRQ/ nie jest wykorzystywane. Może jednak zostać przyłączone do mikroprocesora (konstrukcja typu „otwarty dren" pozwala na galwaniczne sumowanie z innymi sygnałami przerwań).

Częstotliwość sygnału na wejściach RxC i TxC ustalona została na ok. 20 kHz, co w zależności od zaprogramowania wewnętrznego dzielnika odpowiada transmisji z szybkością ok. 19 200 bodów (podział przez 1), 1200 bd (podział przez 16) lub 300 bd (podział przez 64). Sygnał RxC i TxC wy-



Rys. 2. Przebiegi czasowe dla wybranych sygnałów



Rys. 1. Schemat ideowy układu z mikroprocesorem 6800

twarzany jest z sygnału MCLK o częstotliwości 1 MHz w dzielnikach U14 (7490), dzielącym przez 5 i U15 (7490) dzielącym przez 10 ze współczynnikiem wypełnienia 1/2.

W zależności od typu używanego terminala linie RxD i TxD oraz ewentualnie RTS/ i CTS/ należy zaopatrzyć w odpowiednie nadajniki-odbiorniki linii. Dla standardu RS 232C będą to pary MC1488/1489 firmy MOTOROLA bądź SN75150/SN75154 firmy TEXAS INSTRUMENTS.

Dekoder adresowy, zbudowany z U9 (74LS138) i połówki U7 (74S139) wybiera układ ACIA pod adresami 8 i 9 strony układów we-wy.

W bardzo obszernej literaturze omawiającej rodzinę M6800 pozycje krajowe są niestety nieliczne i nieaktualne. Pełną informację techniczną dotyczącą MC6800 i MC6850 można znaleźć w skrypcie Politechniki Warszawskiej: C. Stępień i in.: „Układy mikroprocesorowe serii INTEL 8080, MOTOROLA 6800 i Am 2900". Podstawowe pozycje firmowe to: „M6800 Microprocessor Applications Manual", „M6800 Microprocessor Programming Manual" oraz oczywiście katalog [1].

JERZY ORKISZEWSKI
Warszawa

[1]) Obszernym wyborem katalogów firmy MOTOROLA dysponuje czytelnia PIE-BOINTE, Warszawa, ul. Marynarska 10 (przyp. AJP)

Wykorzystanie mikrokomputera jako notatnika ma swoje wady i zalety. O ile notes można zawsze mieć pod ręką, to ZX 81 wraz z telewizorem wymaga wcale niemałej torby. Jeżeli jednak korzystamy z często zmieniającego się zbioru informacji, to notes wkrótce stanie się mało czytelny. Polecamy więc program osobom prowadzącym „burzliwą" działalność ale... w jednym miejscu.

Przytoczona przez autora artykułu wersja programu dedykowana jest... pośrednikom w handlu nieruchomościami. Oczywiście bez trudu można ją przystosować do informacji dowolnej natury. Sądzimy, że pokazanie pewnego konkretnego zastosowania ułatwi czytelnikom zrozumienie sposobu funkcjonowania programu.

SPREADSHEET dla ZX 81

Nazwa „SPREADSHEET" odnosi się do pewnego rodzaju programów, umożliwiających przeglądanie większej przestrzeni niż tylko tej ograniczonej do powierzchni ekranu. W tym przypadku ekran jest jak gdyby okienkiem, przez które oglądamy fragment większej całości.

Taka technika jest stosunkowo wygodna przy tworzeniu wieloformatowych tabel, zawierających, jak w tym przypadku, 100 kolumn i 60 wierszy. Dla tego rodzaju programów w Polsce stosuje się czasem nazwę „tablplan".

```
  1 REM 0.....
.....1.....
.....2.....
.....3.....
.....4.....
.....5.....
.....6.....
.....7.....
.....8.....
.....9.....
.....0.....
.....1.....
.....2.....
```

Wydruk 1. Linia „1" programu

Pierwszym krokiem w tworzeniu przedstawianego programu jest umieszczenie linii: 1 REM i 395 dowolnych znaków — w naszym przypadku są to kropki. Poszczególne wiersze tego wydruku zostały ponumerowane dla ułatwienia liczenia (wydruk 1). Zanim zaczniemy wystukiwać owe rzędy kropek, przełączmy komputer na tryb pracy FAST, co zaoszczędzi nam sporo czasu. Po wystukaniu wymaganej liczby kropek (może ich być więcej — nigdy mniej) należy komputer przełączyć na SLOW i dla uchronienia się przed skutkami ewentualnych późniejszych błędów — przepisać na kasetę efekt dotychczasowej pracy. Nie kasując linii 1 REM, wprowadzamy program przedstawiony na wydruku 2 i od tej chwili jesteśmy gotowi do zapełnienia poszczególnych komórek pamięci liczbami przedstawionymi na wydruku 3, poczynając od adresu 16 514. Przedtem jednak kilka słów o wprowadzonym przed chwilą do pamięci „assemblerze" [1]). Jest on tak skonstruowany, że liczby wprowadzane do pamięci ukazują się na ekranie monitora po sześć w wierszu, co bardzo ułatwia kontrolę prawidłowości prowadzonej pracy. Należy tu podkreślić, że wprowadzanie do pamięci poszczególnych liczb jest najbardziej odpowiedzialnym etapem tworzenia programów maszynowych, gdyż właśnie w tej fazie powstaje najwięcej błędów.

```
9800 LET m=16514
9810 LET n=0
9820 INPUT a
9830 IF a>300 THEN GO TO 9920
9840 POKE m,a
9850 PRINT AT 20,m;a
9860 LET m=m+1
9870 LET n=n+5
9880 IF n<26 THEN GO TO 9820
9890 LET n=0
9900 REM SCROLL
9910 GO TO 9820
9920 IF a>555 THEN GO TO 9999
9930 IF a=555 THEN GO TO 9980
9940 LET N=N-1
9950 IF n<26 THEN LET n=n-5
9960 PRINT AT 20,n;" "
9970 GO TO 9820
9980 SAVE "ASSEMBLER"
9990 GO TO 9820
9999 STOP
```

Wydruk 2. Program umożliwiający wprowadzenie kodu maszynowego

Proponowany tu „assembler" zawiera pewne ułatwienia w wykonywaniu tej niewdzięcznej pracy. I tak: wprowadzenie liczby 333 [2]) umożliwia nam poprawienie błędnie wprowadzonej ostatniej liczby. Wybrano liczbę 333, gdyż są to trzy uderzenia w ten sam klawisz. Liczba 555 jest równoważna instrukcji SAVE, która powoduje zapisanie na taśmie wszystkich dotychczas wprowadzonych liczb. Można jedynie radzić by pomocniczą liczbę 555 wprowadzać stosunkowo często, zapisując wynik naszej pracy nawet co trzy, cztery wiersze. Tak częste zapisywanie na kasecie programów cząstkowych wcale nie jest przesadą — będzie się można o tym przekonać przy wprowadzaniu programu.

Dla łatwiejszej orientacji długie kolumny liczb podzielono na trzy grupy. Po wprowadzeniu ostatniej liczby należy nacisnąć trzy dziewiątki czyli 999, co jest równoznaczne instrukcji STOP zatrzymującej pracę „assemblera".

Zapewne nurtować nas będzie teraz pytanie: czy udało się nam wprowadzić wszystkie liczby bez pomyłek? Odpo-

[1]) Program przedstawiony na wydruku 2 został przez autora żartobliwie nazwany „assembler". Chociaż kompletuje on kod maszynowy programu, to jednak jest daleką namiastką programów typu asembler wykorzystywanych do translacji źródłowej (w postaci symbolicznej) wersji programu na binarną. Ponieważ autor użył tej nazwy również na wydruku — zachowaliśmy ją w tekście artykułu. (przypis AJP)

[2]) Największa liczba będąca kodem programu wynosi 255 (przypis AJP)

wiedź uzyskamy przez zsumowanie zawartości poszczególnych komórek, co najlepiej można zrobić przy użyciu następującego programiku:

```
9995 LET M=0
9996 FOR N=16514 TO 16870
9997 LET M=M+PEEK N
9998 NEXT N
9999 PRINT M
```

```
0    0    0    0    0    0
0    0    0    0    62   8
14   0    33   130  64   113
35   61   254  0    32   249
42   16   64   126  254  198
40   3    35   24   248  35
126  254  117  40   3    35
24   239  62   7    35   61
254  0    32   250  34   132
64   58   134  64   42   132
64   1    100  0    254  0
40   5    237  74   61   24
247  34   132  64   58   135
64   42   132  64   254  0
40   4    35   61   24   248
34   132  64   42   12   64
1    166  0    237  74   34
130  64   205  187  2    124
198  2    56   9    68   77
205  189  7    6    0    78

121  50   137  64   1    0
12   11   120  177  32   251
58   137  64   254  35   32
14   58   134  64   254  50
40   7    58   134  64   60
50   134  64   58   137  64
254  34   32   14   58   134
64   254  0    40   7    58
134  64   61   50   134  64
58   135  64   254  0    40
14   58   137  64   254  36
32   7    58   135  64   61
50   135  64   58   135  64
254  68   40   14   58   137
64   254  33   32   7    58
135  64   60   50   135  64
62   10   50   136  64   42
16   64   126  254  217  40
3    35   24   248  35   126
254  103  40   3    35   24

239  35   35   35   35   35
58   135  64   254  0    40
4    35   61   24   248  34
138  64   42   12   64   1
100  0    237  74   237  91
138  64   14   33   26   119
13   121  61   35   19   254
0    40   2    24   243  42
130  64   237  91   132  64
14   33   58   136  64   254
10   32   5    26   198  128
24   1    26   119  13   121
61   35   19   254  0    40
2    24   231  35   62   68
19   61   254  0    32   250
58   136  64   61   50   136
64   254  0    194  168  65
58   137  64   254  50   40
7    254  40   40   3    195
154  64   201  27   27   27
```

Wydruk 3. Kod maszynowy programu

Program ten uruchamiamy instrukcją RUN 9995. Jeżeli otrzymana suma wyniesie 29 192 — możemy przystąpić do następnego etapu. Jeżeli wynik nie jest zgodny z podaną

```
  5 LET A=1
 10 DIM T$(100)
 20 DIM A$(60,100)
 30 LET T$="TYPE PRICE 1ST MOR
TGAGE RATE LOAN           ADDRESS
              AGENCY PHONE"
 40 FOR N=1 TO 60
 50 FAST
 60 LET A$(N)="    :$      :$
      :    :    :
           :         :        :
                   :"
 70 LET A$(N,2 TO 3)=STR$ N
 80 NEXT N
 90 SLOW
105 CLS
110 PRINT AT 5,11;"M E N U";AT
7,8;"(ENTER CHOICE)";AT 10,5;"1.
 ENTR RECORD.";AT 12,5;"2. VIEV
SPREADSHEET."
120 IF INKEY$="1" THEN GOTO 200
130 IF INKEY$="2" THEN GOTO 150
140 GOTO 120
150 CLS
160 PRINT "  REAL ESTATE SPREAD
SHEET";AT 16,0;"
                ";TAB 6;"PRESS ""
M"" FOR MENU."
170 RAND USR 16524
180 IF PEEK 16521=50 THEN GOTO
100
182 IF PEEK 16521=40 THEN GOTO
186
184 GOTO 100
186 PRINT AT 18,4;"ENTER CORREC
TED ADDRESS"
188 INPUT A$(PEEK 16518+1,37 TO
67)
190 RAND USR 16607
192 GOTO 180
200 CLS
210 PRINT "ENTER RECORD INFORMA
TION"
220 PRINT ,,"1. TYPE? (4 LETTER
S)"
230 INPUT A$(A,1 TO 4)
240 PRINT "2. PRICE?"
250 INPUT X
260 LET A$(A,13-LEN STR$ X TO 1
2)=STR$ X
270 PRINT "3. FIRST MORTGAGE PR
ICE?"
280 INPUT X
290 LET A$(A,21-LEN STR$ X TO 2
0)=STR$ X
300 PRINT "4. 1ST MTG. LOAN RAT
E?"
310 INPUT X
320 LET A$(A,31-LEN STR$ X TO 3
0)=STR$ X
330 PRINT "5. TYPE OF LOAN (VA-
FHA-CONV.)?"
340 INPUT A$(A,32 TO 35)
350 PRINT "6. PROPERTY ADDRESS?
"
360 INPUT A$(A,37 TO 67)
370 PRINT "7. CITY?"
380 INPUT A$(A,69 TO 83)
390 PRINT "8. AGENCY OFFERING P
ROPERTY?"
400 INPUT A$(A,85 TO 91)
410 PRINT "9. AGENCIES PHONE NU
MBER?"
420 INPUT A$(A,93 TO 100)
430 LET A=A+1
440 GOTO 100
450 SAVE "SPREADSHEET"
```

Wydruk 4. Program „Spreadsheet" (część napisana w języku BASIC)

liczbą, nie należy podejrzewać pomyłki zecera, gdyż wydruk 3 jest fotokopią wydruku działającego programu. Teraz może nam się przydać zapis na taśmie wprowadzanych liczb. Gdy uda się nam uzyskać poprawną sumę 29 192 możemy wprowadzić z klawiatury treść wydruku 4, tworzącego „szatę graficzną" naszego programu. Układ, rozmiar i treść można oczywiście dowolnie zmieniać w celu dostosowania programu do potrzeb i gustów użytkownika. Przed uruchomieniem programu należy wykasować wszystkie linie od 9000 do 9999, gdyż nie będą już nam potrzebne.

Po uruchomieniu, program zgłasza się planszą tytułową, proponując nam dwupozycyjne menu:
klawisz 1 — wprowadzenie danych
klawisz 2 — wyświetlenie tabeli.

Po naciśnięciu 1 ukazuje się treść rubryk, które kolejno zapełniamy, udzielając odpowiedzi na stawiane pytania. Korzystanie z zawartości tabeli możliwe jest po naciśnięciu klawisza 2. Naciskanie odpowiednich strzałek (bez SHIFT) umożliwia wybranie interesującego nas fragmentu tabeli.

W okienku ukazuje się 10 wierszy. Stosownie do potrzeb możemy okienko zwiększyć lub zmniejszyć (od 1 do 17 wierszy). Gdy zechcemy oglądać np. 15 wierszy, musimy dokonać następujących zmian: POKE 16371,15 oraz POKE 16657,60 — 15. Na tym jednak nie koniec, gdyż naruszona została struktura tablicy. Musimy także wprowadzić liczbę wierszy (tj. 15) do komórki o adresie 16814 — a więc POKE 16814,15.

Program ten zawiera też możliwość dokonywania zmian w rubryce „ADDRESS", dzięki istnieniu w programie wierszy od 180 do 192. Po naciśnięciu „C" ukazuje się napis „ENTER CORRECTED ADDRESS". Po wprowadzeniu nowego adresu zostanie on wpisany do pierwszego z aktualnie wyświetlanych wierszy. Dla wyróżnienia, pierwszy wiersz jest wyświetlany w postaci negatywowej.

W komórce o adresie 16640 umieszczona jest liczba 12, regulująca prędkość poruszania się „okienka" po tablicy. Liczba większa niż 12 spowolni przesuwanie się okienka, zaś liczba mniejsza przyśpieszy ruch kolumn i wierszy na ekranie. Umożliwia to dostosowanie prędkości przesuwania się okienka do własnych wymagań.

W przestawionej wersji programu tablica składa się z 60 wierszy i 100 kolumn. Poniższa tabelka podaje niezbędne zmiany jakie należy dokonać w celu uzyskania tablic o wymiarach: 50×115, 80×100 i 100×100.

Zmiany jakie należy wykonać w programie dla uzyskania tabeli o innych wymiarach, niż podane w przykładzie

Operacja	50×115	80×100	100×100
A$ DIM	(50, 115)	(80, 100)	(100, 100)
T$ DIM	(115)	(100)	100)
POKE 16552,	123	69	21
POKE 16576,	115	100	100
POKE 16657,	40	70	90
POKE 16713,	83	68	68
POKE 16749,	118	103	103
POKE 16837,	83	68	68

Po dokonaniu wskazanych w tabeli zmian w liniach 10 i 20 programu oraz po zmianie zawartości wymienionych komórek pamięci, musimy pamiętać jeszcze o przystosowaniu wierszy 40 i 60 do nowych rozmiarów tabeli.
Program przeznaczony jest dla ZX 81 z pamięcią 16 KB.

WITOLD MAJEWSKI
Wrocław

Niektórzy ze współpracowników mikroKLANU przeglądając jeszcze w maszynopisie tekst artykułu o sprzętowej realizacji poszerzenia możliwości graficznych ZX 81 mocno kręcili głowami: po co „wpuszczać" Czytelników w krajanie komputera, jeżeli istnieje gotowy program spełniający podobną funkcję w sposób „bezkrwawy".

Przeważył jednak argument, że skalpel i lutownica znajdują się praktycznie w zasięgu każdego posiadacza ZX 81, natomiast wspomniany program — niekoniecznie.

Okazuje się jednak, że wykonana przeróbka pozwala również... rozszerzyć możliwości programu „High Resolution Graphic" (ang. grafika z Dużą Rozdzielczością).

Jeszcze o możliwościach graficznych ZX 81 16 KB

Opisana w 14 mikroKLANIE przeróbka komputera ZX 81 16 KB umożliwiła wykonanie pierwszego kroku w kierunku poprawy jego możliwości graficznych. Dzięki stosunkowo prostej zmianie konfiguracji połączeń wewnętrznych udało się stworzyć możliwość przechowywania tablicy znaków w pamięci RAM, co pozwoliło na deklarację dowolnego ich zestawu. Tym, którzy zdecydowali się wykonać te zmiany, proponuję kolejne — tym razem już tylko programowe — rozwiązanie, pozwalające na dalsze rozszerzenie możliwości graficznych komputera. Rozwiązanie to wykorzystuje program „High Resolution Graphic" (dostępny w kraju dzięki rozwijającemu się prywatnemu importowi oprogramowania i sprzętu komputerowego).

Program HRG umożliwia uzyskanie rozdzielczości 192×256 z wyłączeniem niektórych punktów. Uzyskanie tak dużej poprawy rozdzielczości stało się możliwe dzięki przejęciu sterowania wyświetlaniem na ekranie i zmianie wymiaru każdego ze znaków z 8×8 na 1×8. Dodatkowo, program zmienia położenie tablicy znaków w ROM i z nowej tablicy wykorzystuje co ósmy bajt, jako znak graficzny. Pamięć obrazu zajmuje 6,2 KB pamięci i przechowywane są w niej kody znaków umożliwiające programowi odnalezienie położenia znaku w tablicy.

Istnieją jednak pewne ograniczenia programu. Brakuje niestety znaków symbolizujących niektóre pojedyncze punkty. Nie są też możliwe wszystkie kombinacje położenia punktów w znaku. Ograniczenia te można usunąć wprowadzając do programu HRG zmiany, wykorzystujące możliwości przerobionego komputera. Zasadą tych zmian jest przesunięcie tablicy znaków do RAM, zadeklarowanie odstępu i ośmiu znaków oznaczających kolejne punkty (tzn. wpisanie do pamięci co ósmego bajtu wartości: 0, 1, 2, 4, 8, 16, 32, 64, 128) oraz deklarowanie nowych znaków w miarę potrzeb wykreślonego rysunku. Oczywiście deklarowaniem tym zajmuje się specjalny podprogram, który najpierw sprawdza czy wymagany znak znajduje się już w pamięci i ewentualnie deklaruje go. Jak wykazała praktyka na 64 możliwe znaki, wykorzystanych zostaje:
17 znaków — przy wykreślaniu kwadratów i prostokątów
23 znaki — przy wykreślaniu funkcji sinus i cosinus
35 znaków — przy wykreślaniu okręgu.
Czas wykonywania rysunków (pod warunkiem napisania podprogramu deklarującego w kodzie maszynowym) jest w przybliżeniu taki sam, jak w przypadku oryginalnego programu.

Efekty? Sprawdźcie sami.

JACEK PSZONA
Warszawa

W poprzednim mikroKLANIE opisaliśmy algorytm definiowania dodatkowych komend rozszerzających BASIC w ZX SPECTRUM. W tej części artykułu publikujemy przykład konkretnej realizacji tego algorytmu. Wydruk procedury napisanej w języku ASSEMBLER Z80 można z łatwością wykorzystać jako „szkielet" własnych, być może o wiele ciekawszych niż prezentowana w przykładzie, instrukcji. Zachęcamy do prezentacji ich na naszych łamach.

Jak rozszerzyć BASIC ZX SPECTRUM (2)

Jak wspomniano w poprzedniej części tekstu, dla przejęcia sterowania od INTERPRETERA języka BASIC należy w programie umieścić pewien błędny zapis. W przykładzie wykorzystano klucz **SCREEN$**, który normalnie wykorzystywany jest w instrukcjach **LOAD** i **SAVE** dla odczytu i zapisu na taśmie obrazu z ekranu oraz dla odczytania znaku w określonym miejscu ekranu. Inne użycie tego klucza powoduje generację komunikatu błędu i przerwania realizacji programu.

Nowa instrukcja ma postać:

SCREEN$ p,k

gdzie p — to numer odpowiadający nowo definiowanemu kolorowi tła, natomiast k — to numer odpowiadający numerowi koloru nowo definiowanej kreski. Wykonanie instrukcji powoduje zmianę kolorów na całym ekranie. Za wyborem klucza SCREEN$ przemawia prosty sposób sprawdzania syntaktyki (poprawnego zapisu) nowej instrukcji, gdyż daje on pojedynczy kod przechowywany w pamięci obok argumentów. Analogiczną rolę mogą spełniać klucze od RND (kod: 165) do STEP (kod: 205).

Program przedstawiony na wydruku 1 może bez żadnych modyfikacji być wykorzystany zarówno dla wersji 16K jak i 48K. W instrukcjach DATA zawarto kod maszynowy. Uruchomienie programu powoduje umieszczenie kodu za pozycją wskazywaną przez adres przechowywany w zmiennej **RAMTOP.** Dla wersji 16K jest to lokacja o adresie 32401, natomiast dla wersji 48K jest to pozycja o adresie 65101. W trakcie przepisywania kodu sprawdzana jest suma wszystkich danych (powinna ona wynosić 16918). Jeżeli wprowadzony kod nie jest poprawny, przerywana jest realizacja programu. Wskazane jest jednak zarejestrowanie programu na taśmie przed pierwszą próbą uruchomienia, gdyż złośliwy błąd może dać prawidłową sumę kontrolną i równocześnie spowodować blokadę komputera. Jeżeli stracimy kontrolę nad programem, pozostanie jedynie wyłączenie i włączenie komputera — a wtedy mozolnie wprowadzone dane przepadną.

W drugiej części programu demonstrowany jest przykład ilustrujący sposób działania nowej instrukcji **SCREEN$.**

```
  1 REM ********************************
  2 REM ** "SCREEN$" - jako komenda **
  3 REM ********************************
 10 CLEAR 32400+32700*(PEEK 23733=255)
 15 REM
 16 REM ustawianie RAMTOP zaleznie od wersji (16K lub 48K)
 17 REM
 20 LET start=1+PEEK 23730+256*PEEK 23731
 25 REM
 26 REM Kod bedzie wprowadzony za RAMTOP
 27 REM
 30 RESTORE : LET sum=0
 40 FOR j=start TO start+157
 50 READ n: POKE j,n
 60 LET sum=sum+n
 70 NEXT j
 80 IF sum<>16918 THEN PRINT "Blad sumy kontrolnej": STOP
 90 REM ------------------------------------------------
410 DATA 58,58,92,254,11,40,35,253,203,1,126,32,8,42,178,92,35
411 DATA 229,195,183,18,205,3,19
412 DATA 253,54,0,255,42,89,92,205,167,17,42,178,92,35,229,195
413 DATA 180,18,42,93,92,43,126,254
414 DATA 170,32,212,253,54,0,255,253,54,38,0,205,122,28,253
415 DATA 203,0,126,40,195,254,13,40,8
416 DATA 253,54,0,11,254,58,32,183,253,54,0,255,253,203,1,126
417 DATA 32,12,42,178,92,35,229,33
418 DATA 183,18,229,195,118,27,42,141,92,34,143,92,205,148,30
419 DATA 254,8,40,5,87,55,205,53,34
420 DATA 205,148,30,254,8,40,5,87,167,205,53,34,205,173,28,58
421 DATA 141,92,33,0,88,17,1,88
422 DATA 1,255,2,119,237,176,42,178,92,35,229,195,118,27
430 REM ------------------------------------------------
495 REM
496 REM Ustawianie adresu "nowej" obslugi bledu
497 REM
500 LET errsp=PEEK 23613+256*PEEK 23614
510 POKE errsp,start-256*INT (start/256)
520 POKE errsp+1,INT (start/256)
601 REM ************************************************
602 REM ************************************************
603 REM * Nalezy wykonac dotychczas wprowadzone linie  *
604 REM * dla sprawdzenia poprawnosci wprowadzonego kodu *
605 REM * przed wprowadzeniem pozostalych              *
606 REM ************************************************
607 REM ************************************************
608 REM
700 CLS : LIST 740:SCREEN$ 5,1
710 FOR j=1 TO 3
720 PRINT '"CAPSSHIFT & BREAK zatrzymuja"'"          Program"
730 NEXT j
740 FOR j=0 TO 7
750 FOR k=0 TO 7
760 PAUSE 20
770 IF j=k THEN NEXT k
900 SCREEN$ j,k
910 NEXT k
920 NEXT j
930 GO TO 740
1000 STOP
```

Wydruk 1.

```
          ORG  RAMTOP+1     ;Procedura jest relokowalna, musi jednak
                            ;byc ladowana zaraz ponad RAMTOP
START     LD   A,(26610)    ;Czy kod bledu = 11 (Nonsense in BASIC)
          CP   11
          JR   NZ,NONSENSE
ERROR     BIT  7,(IY+1)     ;Bit 7 zmiennej FLAGS jest ustawiony
          JR   NZ,RUNERROR  ;tylko w trybie realizacji programu
SYNTXER   LD   HL,(23730)   ;Blad syntaktyczny - START = RAMTOP+1
          INC  HL           ;START odkladany na stos - gotowy na
          PUSH HL           ;nastepny blad
          JP   12B4H        ;Powrot do ROM
RUNERR    CALL 1303H        ;Blad realizacji - wypisz komunikat
          LD   (IY+0),255   ;Zerowanie numeru bledu
          LD   HL,(23641)   ;Usuniecie zapisow zmiennoprzecinkowych
          CALL 11A7H        ;z obszaru edytowanego przed
          LD   HL,(23730)   ;sprawdzeniem syntaktyki
          INC  HL           ;START odkladany na stos
          PUSH HL
          JP   12B4H        ;Powrot do ROM
NONSENS   LD   HL,(23645)   ;W CH ADD znajduje sie adres do ktorego
          DEC  HL           ;dotarl interpreter. Pobranie znaku,
          LD   A,(HL)       ;ktory byl przyczyna bledu
          CP   170          ;Czy to byl SCREEN$ ?
          JR   NZ,ERROR     ;Skok pod ERROR jezeli nie
          LD   (IY+0),255   ;Zerowanie ERR NR i X PTR
          LD   (IY+38),0
          CALL 1C7AH        ;Sprawdzenie czy nastepne w linii sa
          BIT  7,(IY+0)     ;dwie liczby oddzielone przecinkiem;
          JR   Z,ERROR      ;jezeli nie to ERR NR wskazuje blad.. CH-
          CP   13           ;ADD jest przesuwany a do A wprowadzany
          JR   Z,OK         ;jest nastepny znak (ENTER lub dwukropek
          LD   (IY+0),11    ;gdyz innaczej blad: "Nonsense in
          CP   58           ;BASIC")
          JR   NZ,ERROR
OK        LD   (IY+0),255   ;Jezeli syntaktyka jest poprawna . to
                            ;zerowanie ERR NR
          BIT  7,(IY+1)     ;Jezeli tryb realizacji to "nowa"
          JR   NZ,DO-IT     ;instrukcja moze teraz byc wykonana
          LD   HL,(23730)   ;W innym przypadku adresy START i 12B7H
          INC  HL           ;sa odkladane na stos i ma miejsce
          PUSH HL           ;powrot do ROM-u
          LD   HL,12B7H
          PUSH HL
          JP   1B76H
DO-IT     LD   HL,(23693)   ;Realizacja. Na wstepie kolory "stale"
          LD   (23695),HL   ;kopiowane sa do "tymczasowych". Kolor
          CALL 1E94H        ;kreski pobierany jest ze stosu. Jesli
          CP   8            ;jest = 8 to bez zmian, jesli nie to
          JR   Z,PAPER      ;procedura z ROM-u wykorzystywana jest
          LD   D,A          ;do zmiany ATTR T kreski
          SCF
          CALL 2235H
PAPER     CALL 1E94H        ;Kolor papieru pobierany jest ze stosu i
          CP   8            ;jezeli nie jest = 8 to powtornie
          JR   Z,OUT        ;wywolywana jest procedura z ROM-u
          LD   D,A
          AND  A
          CALL 2235H
OUT       CALL 1CADH        ;"Tymczasowe" kolory przechodza na
          LD   A,(23693)    ;stale. Rozkaz LDIR wykorzystywany jest
          LD   HL,5800H     ;do ustawienia atrybutow koloru takich
          LD   DE,5801H     ;samych jak ATTR P.
          LD   BC,2FFH
          LD   (HL),A
          LDIR
          LD   HL,(23730)   ;START odkladany jest na stos
          INC  HL           ;przygotowujac obsluge nastepnego bledu.
          PUSH HL
          JP   1B76H        ;Powrot do ROM-u.
```

Wydruk 2.

Jeżeli program funkcjonuje prawidłowo, powinny mieć miejsce sekwencyjne zmiany kolorów tła i kreski. Realizację programu można przerwać w dowolnym momencie, naciskając równocześnie klawisze **BREAK** i **CAPS SHIFT.**

Użyteczne procedury w ROM

Adres procedury (szesnastkowo)	Wykonywane działania
18H	RST 18H ładuje kolejny znak wskazywany przez **CH ADD** do akumulatora. Znaki niedrukowalne (np. kody kolorów) są ignorowane, a wskaźnik w **CH ADD** jest przesuwany aż do odczytania „normalnego" znaku.
20H	RST 20H powoduje inkrementację wskaźnika w **CH ADD** i wprowadzenie kolejnego znaku do akumulatora.
1C82H	Procedura wyznacza wartość wyrażenia numerycznego i jeżeli komputer pracuje w trybie realizacji wpisuje ją na stos kalkulatora. Przed wywołaniem procedury zmienna **CH ADD** powinna wskazywać adres pierwszego znaku wyrażenia. Po wyjściu z procedury **CH ADD** wskazuje pierwszy znak za wyrażeniem. Jeżeli wyrażenie było numeryczne, to bit 6 w zmiennej **FLAGS** jest ustawiony na 1.
1C7AH	Procedura wyznacza wartości dwóch wyrażeń numerycznych, rozdzielonych przecinkiem i wpisuje je na stos kalkulatora. **CH ADD** i **FLAGS** — identycznie jak dla 1C82H.
1E94H	Procedura pobiera liczbę ze stosu kalkulatora i wpisuje ją do akumulatora. Liczba musi być dodatnia i mniejsza od 256.
1E99H	Pobranie liczby ze stosu kalkulatora i wprowadzenie jej do pary rejestrów BC. Liczba musi być dodatnia i mniejsza od 65536.

Na wydruku 2 przedstawiono postać symboliczną procedury zapisanej w języku ASSEMBLER Z80. Realizuje ona algorytm przedstawiony w pierwszej części artykułu.

Zamiast klucza produkującego pojedynczy kod, nowa instrukcja może być też zapisywana w sposób bezpośredni (tzn. litera po literze). Aby możliwe było wpisanie takiej instrukcji należy ją poprzedzić znacznikiem powodującym zmianę trybu wprowadzania przez komputer (z kursora K na L). Znacznikiem takim może być np. *. Syntaktyka (poprawność) takiej instrukcji również musi być sprawdzana litera po literze. Wykrycie niezgodności zapisu w programie ze wzorcem powoduje powrót do standardowej procedury sygnalizującej błąd. Przeglądanie zapisu instrukcji najprościej można zrealizować wykorzystując wskaźnik **CH ADD** (adres: 23645) i zawarte w ROM procedury RST 18H i RST 20H. Adresy użytecznych procedur z firmowego ROM przedstawiono w tabeli. Procedura sprawdzająca syntaktykę powinna również kontrolować czy po parametrach instrukcji podano znacznik końca (ENTER lub dwukropek). Po zakończeniu sprawdzania adres zapisany w **CH ADD** powinien wskazywać znacznik końca — w innym przypadku zostanie zakłócona rutynowa praca INTERPRETERA języka BASIC.

Po każdym wywołaniu procedura pozostawia na stosie wartości pozwalające na ponowne odwołanie się do nowej instrukcji. Uruchomienie programu przez dyrektywę RUN zeruje jednak stos i wprowadza „normalny" adres obsługi błędu. Dlatego też przed pierwszym odwołaniem do nowej instrukcji należy wykonać instrukcję POKE odpowiednie zmiany, tak jak to pokazano w przykładowym programie zamieszczonym na wydruku 1.

MAREK GÓRECKI
ANDRZEJ J. PIOTROWSKI
Warszawa

W tym numerze mikroKLANU prezentujemy ostatni z regularnych wykładów Akademii, publikując informatyczną odmianę praw Murphy'ego. Są one niejako kwintesencją omawianych do tej pory reguł. Chociaż wiele z nich brzmi humorystycznie, to jednak poważne ich traktowanie może oszczędzić wielu rozczarowań i kłopotów.

Ponawiamy propozycję publikowania nadesłanych nam przez Czytelników krótkich procedur w języku BASIC. Liczymy naturalnie, że będą one nie tylko „chodzące", ale też zapisane w zgodzie z propagowanym przez nas eleganckim stylem programowania. I jeszcze jedno: nie planujemy publikowania procedur typu „Metoda obliczania pogłowia trzody chlewnej z wykorzystaniem testu chi-kwadrat" — choćby autor zarzekał się, że spowodujemy tym zniesienie reglamentacji mięsa. Krąg czytelników mikroKLANU stale się poszerza (niezależnie od nakładu limitowanego przez znacznie ważniejsze „czynniki" niż popyt) i musimy dobierać tematy, które zainteresują rozleglejsze grono (a nie tylko autora procedury i jego kolegę z pokoju). Jeżeli jednak ktoś nie jest pewien czy jego program jest interesujący — to mimo wszystko radzimy zaryzykować kilkanaście złotych na znaczek pocztowy.

Akademia mikroKLANU (8)

Kończąc cykl „akademicki" mamy nadzieję, że pozwolił on Czytelnikom mikroKLANU nieco udoskonalić technikę programowania. By Akademii nie zamienić w zbyt szacowny zakład naukowy kończymy kolekcją różnych zasad i przysłów, popularnych wśród programistów. Przyjemnego programowania.

1. Podstawowe prawa arytmetyki komputerowej:
   a) $A=A\pm coś$ na ostatnim miejscu
   b) $A+B\neq B+A$
   c) $A*B\neq B*A$
   d) $A*(B+C)\neq(A*B)+(B*C)$
   e) $A^n\neq A*A*...*A$ (n razy)
   f) odwrócenie operacji nie daje poprzedniej liczby
2. Komputery robią to, co się im każe robić, a nie to, co chcielibyśmy by robiły
3. Nikt nie będzie miał tak dobrej opinii o Twoim programie jak Ty sam.
4. Każdy program zawiera błędy.
5. Pierwsza osoba, która użyje Twego programu, znajdzie w nim błąd.
6. Żaden program nie jest dostatecznie zabezpieczony przed użytkownikami.
7. Żaden program nie działa za pierwszym razem. Bądź ostrożny, jeżeli już przy pierwszej realizacji dostałeś poprawną odpowiedź.
8. Kiedy wszystko inne zawodzi przy poprawianiu programu, wytłumacz problem siedmioletniemu dziecku.
9. Im dłużej trwało poprawianie programu tym pewniejsze, że nie został on bezbłędnie zarejestrowany w pamięci masowej.
10. Każde pół godziny analizy oszczędza dwie godziny programowania.
11. Użyteczność programu jest odwrotnie proporcjonalna do czasu poświęconego na jego przygotowanie.
12. Łatwiej napisać nowy program niż zmodyfikować stary.
13. Każdy problem ma więcej niż jedno rozwiązanie. Wniosek: żadne dwa programy nie są jednakowe.
14. 95 procent czasu poświęca się na rozwiązanie pięciu procent problemu.
15. Większość problemów z komputerem wynika z niedokładnego przestudiowania instrukcji obsługi.
16. Instrukcje są bezużyteczne, jeśli masz praktyczny problem.
17. Zmienne są stałe, stałe są zmienne.
18. Liczby całkowite nie są całkowite.

19. Zmienne zawsze są znane z większą dokładnością niż stałe.
20. Użycie większej liczby cyfr znaczących uczyni Twoje wyniki bardziej wiarygodnymi.
21. Żaden komputer nie będzie miał nigdy wszystkich zalet o jakich marzysz.
22. Zwiększenie pamięci o połowę podwaja możliwości komputera.
23. Zawsze są co najmniej dwa sposoby skrócenia każdego programu.
24. Nigdy nie będziesz miał wystarczającej pojemności pamięci.
25. Każdy program można poprawić.
26. Długo wykonujący się program zawsze „pada" na samym końcu.
27. Wewnątrz każdego dużego programu jest mały program, który chce zniszczyć efekty dużego.
28. Podstawowe prawa programowania:
    a) każdy program gdy już działa, jest niepotrzebny
    b) każdy program kosztuje więcej i przygotowywany jest dłużej niż zaplanowano
    c) jeżeli program jest pożyteczny, trzeba będzie go zmodyfikować
    d) jeżeli program jest bezużyteczny, trzeba go udokumentować.
29. Wszystko powinno być tak proste jak to jest tylko możliwe — ale nie bardziej.
30. Użycie skomplikowanego komputera nie tłumaczy trywialności problemu.

JAKUB TATARKIEWICZ

prowadzi
ANDRZEJ J. PIOTROWSKI
tel. dom.: 48-22-85

Jak grzyby po deszczu powstają w kraju firmy oferujące komputery kompatybilne z IBM PC (najprostsza konfiguracja od 4 mln. zł). Typowa, nowo powstająca firma zbiera zamówienia przez kilkanaście tygodni, tzn. aż... nie wyczerpie się limit danej grupy podatkowej. Realizacja dalszych zamówień wiązałaby się z koniecznością podwyższenia i tak niemałej ceny. Spóźnieni otrzymują terminy na następny rok.

Dysponenci dewiz mogą nabyć taki komputer od firmy ENSCH z Luxemburga (reprezentowana przez MERCOMP) już od 1500 dol. Natomiast zamożniejsi hobbyści składają IBM-y (płytki z Hong Kongu) inwestując poniżej tysiąca dolarów.

Natomiast firma CROMEMCO oferuje na polski rynek mikrokomputery zbudowane na M68000 wraz z systemem UNIX, sztywnym dyskiem (od 50 do 150 MB) i pamięcią RAM od 2 MB za ok. 10 tys. dolarów.

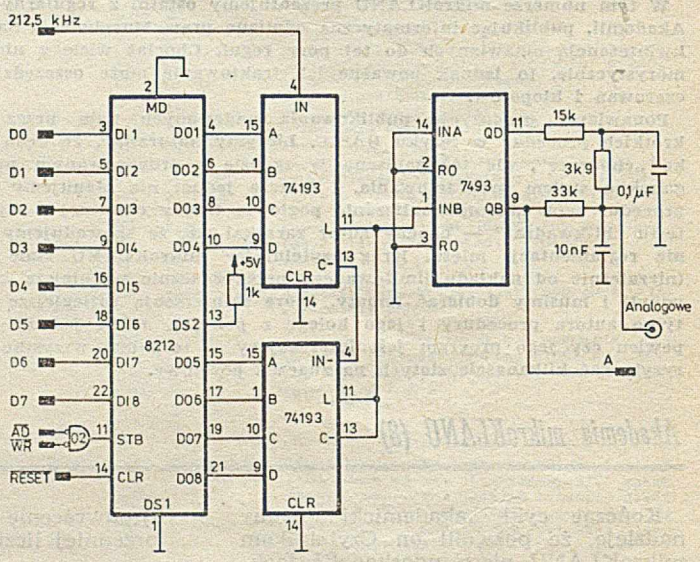
W poprzednim numerze mikroKLANU opisywaliśmy układ scalony, który można traktować niemal jak procesor dźwięków. Układ ten dostępny jest jednak głównie w strefie dolarowej (czasami można go dostać na „perskim") i dlatego też publikujemy obecnie rozwiązanie, które wprawdzie daje znacznie mniejsze możliwości, ale za to można je zrealizować na łatwych do zdobycia elementach.

Prezentowany pomysł jest na tyle prosty, że z nieznacznymi modyfikacjami można go wykorzystać niemal w dowolnej konstrukcji. Zamiast układu 8212 można wykorzystać „wolny" port 8255 a nawet interfejs równoległy do drukarki (jak drukujemy, to nie gramy, i odwrotnie).

Układ syntezy dźwięków

Niemal wszystkie mikrokomputery sprzedawane na rynkach zachodnich zawierają rozbudowane układy dźwiękowe. Służą one do sygnalizacji pracy systemu oraz do „muzycznej oprawy" programów, a czasem nawet do syntezy mowy. Konstrukcje te zawierają specjalizowane układy, procesory dźwiękowe (np. AY-3-8910). Poniżej proponujemy prosty układ syntezy dźwięków, który choć nie ma tak wielu zalet jak układy specjalizowane, jest łatwy do zrealizowania (krajowe elementy) i ma tę właściwość, że otrzymany sygnał jest zbliżony kształtem do sinusoidy.

Układ programowany jest przez wpisanie odpowiedniej wartości do układu 8212 (74S412). Zapis danej z linii D0...D7 następuje po wymuszeniu stanów aktywnych na wejściach WR/ i AD/. Wejście AD/ sterowane jest z dekodera adresów przestrzeni we-wy. Wpisany do portu bajt, określa wartość początkową programowanego dzielnika częstotliwości. Dzielnik zbudowany jest z liczników 74193, liczących „w tył". Liczniki sterowane są sygnałem otrzymywanym z zegara systemowego (2,125 MHz), przez dodatkowy dzielnik 1 : 10. Sygnał „pożyczki" z dzielnika programowanego (c—) wpisuje do liczników informację z bufora 8212, a także steruje licznikiem 7493. Licznik ten generuje przebieg o wypełnieniu 50%, którym sterowany jest układ formowania sygnału zbudowany z elementów RC. Na nich wytwarzany jest sygnał wyjściowy, którego częstotliwość podstawową wyznacza wyjście QD licznika. Wyjście QB wzbogaca go w częstotliwości harmoniczne. W efekcie otrzymujemy sygnał analogowy przypominający sinusoidę. Przy częstotliwości 212,5 kHz sterującej układ programowanego dzielnika, na wyjściu uzyskuje się częstotliwość od ok. 880 Hz. W zakresie słyszalnym umożliwia to uzyskanie, z pewnym przybliżeniem, dwóch oktaw.



Układ ten został zastosowany w komputerze gdzie z wyjścia A pobierany jest sygnał sterujący głośnik kontrolny klawiatury, natomiast sygnał analogowy jest podawany na gniazdo typu „jack" i wzmacniacz zewnętrzny.

TOMASZ PIETRZAK

LITERATURA

[1] Pienkos J., Turczyński J.: Układy scalone TTL w systemach cyfrowych, WKŁ 1980 r.
[2] Informacje techniczne ITE: Układ 74S412.